Deep Equilibrium Models

Shaojie Bai, J. Zico Kolter, Vladlen Koltun

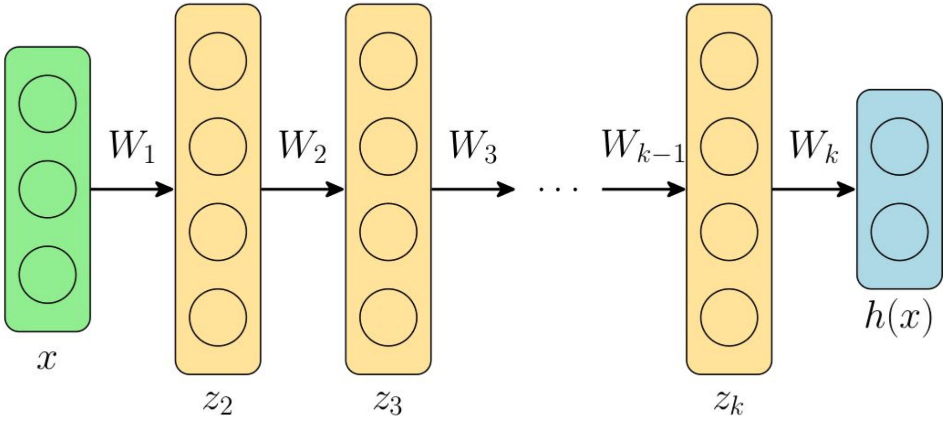
Presented by: Alex Wang, Gary Leung, Sasha Doubov

Agenda

1. Weight tying and infinite depth models
2. Implicit layer formulation
3. Approximation and computational considerations
4. DEQ stacking?
5. Experiments

Motivation

Let's start with a typical deep NN architecture:

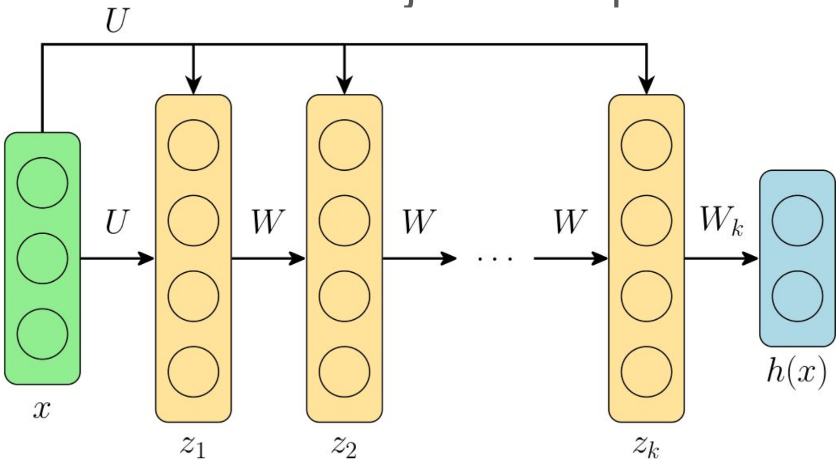


$$
\begin{aligned}
z_1 &= x \\
z_{i+1} &= \sigma(W_i z_i + b_i), \; i = 1, \dots, k-1 \\
h(x) &= W_k z_k + b_k
\end{aligned}
$$

Image courtesy of Deep Implicit Layers Tutorial

Motivation

Weight-Tying: Use the same W and inject the input for each layer



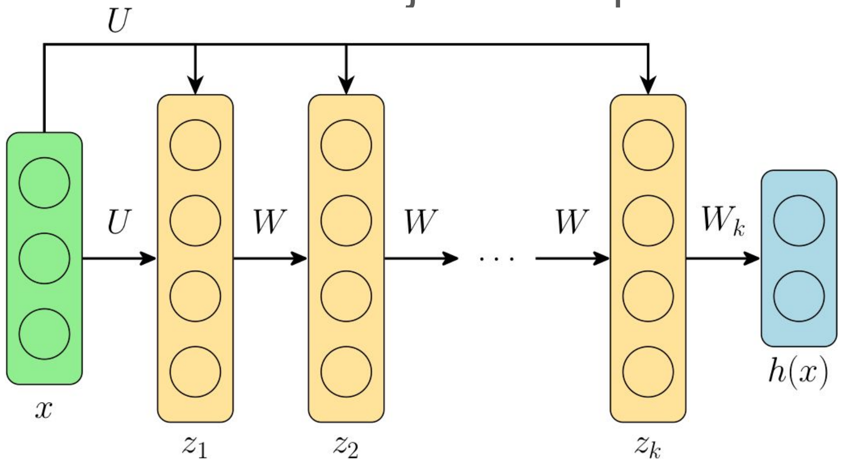
Just as expressive!

$$
\begin{aligned}
z_1 &= 0 \\
z_{i+1} &= \sigma(\boxed{W} z_i + \boxed{U} x + b), \;\; i = 1, \ldots, k-1 \\
h(x) &= W_k z_k + b_k
\end{aligned}
$$

Image courtesy of [Deep Implicit Layers Tutorial]

Motivation

Weight-Tying: Use the same W and inject the input for each layer



Focusing on activation iteration:

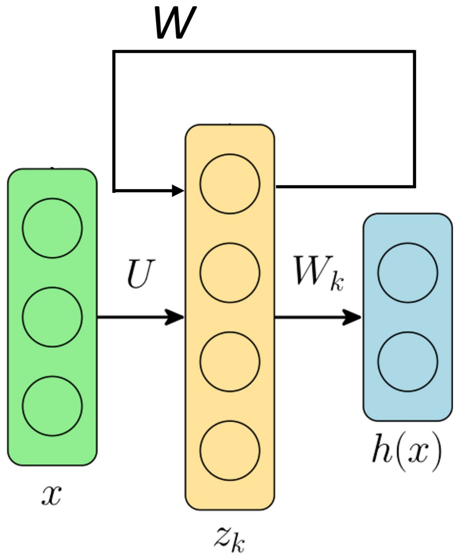
$$z_{i+1} = \sigma(Wz_i + Ux + b), \; i = 1, \ldots, k-1$$

Image courtesy of [Deep Implicit Layers Tutorial]

In the infinite limit, as i -> ∞ (under nice conditions)

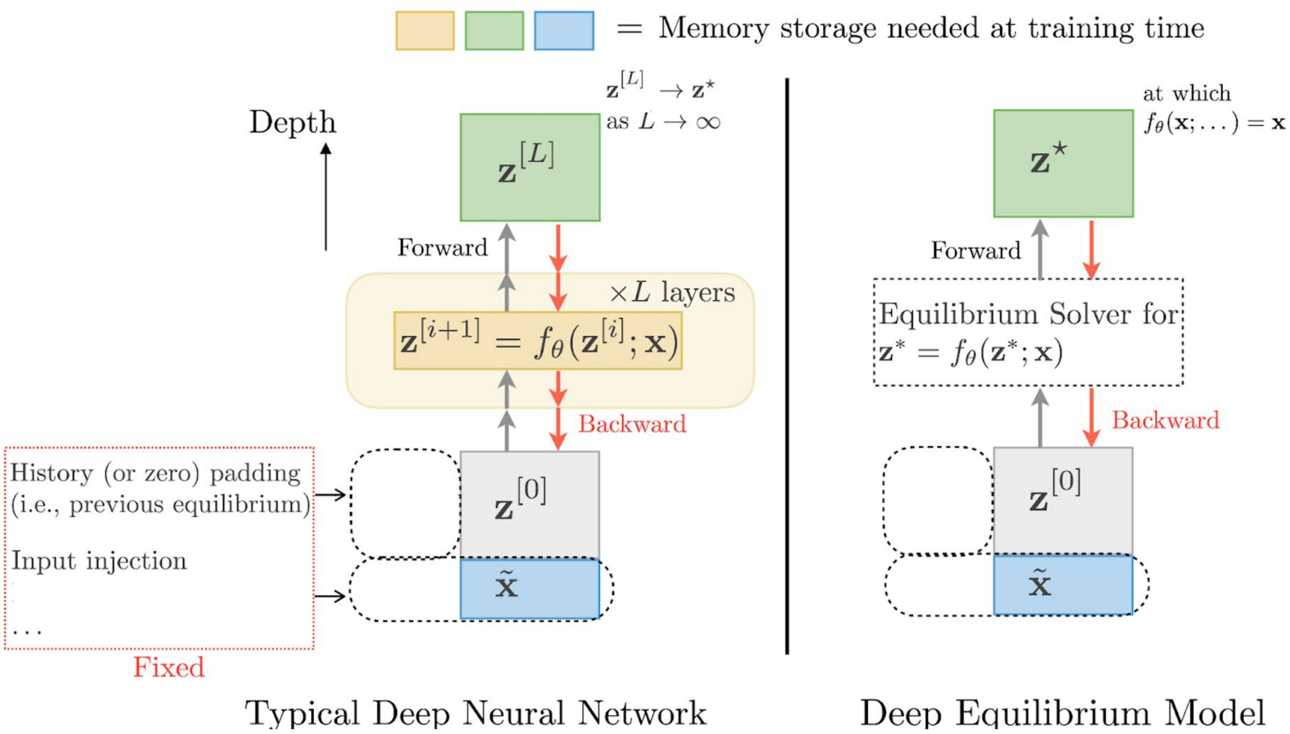
Key insight: The network's activations z* approach a fixed point!

This is a DEQ!



$$\boxed{z^*} = \sigma(W\boxed{z^*} + Ux + b), i \to \infty$$

Deep Equilibrium Model Overview



Implicit vs. Explicit Layers

Explicit Layers:
Typical Neural Network layers, which can directly compute the output and backward pass through backprop

Implicit Layers:
Based on solving solution to some problem, such that **x, z** satisfy some condition

- Arises in naturally in some domains, such as ODEs and *fixed-points*

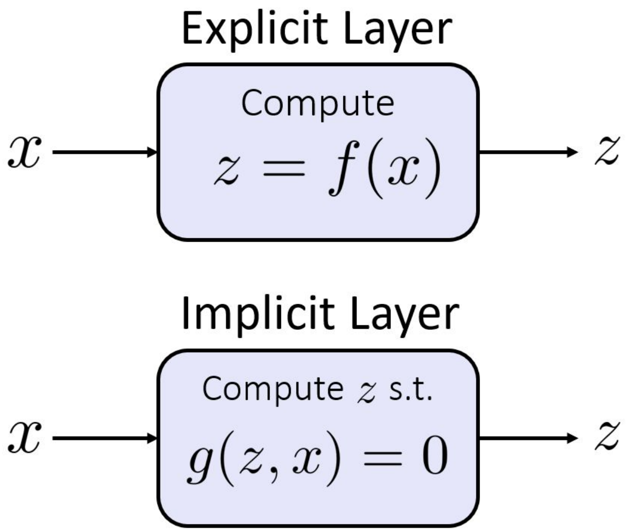


Image courtesy of Deep Implicit Layers Tutorial

Forward Pass

Naive Approach: we could repeatedly apply the function until convergence

$$z^{[i+1]} = f_\theta(z^{[i]}, x) \quad \text{for } i = 0, 1, 2, ...$$

Better Way: Use a root-finding algorithm to find the fixed point

1. Reformulate fixed-point as finding the root:

$$g_\theta(z^*, x) = f_\theta(z^*, x) - z^* \to 0$$

We'll use this notation from here on!

2. Apply generic root-finding algorithm (ex. Newton's method!)

$$\mathrm{z}^* = \mathrm{RootFind}(g_\theta; x)$$

Backward Pass

We need to update the parameters Ɵ in our model, to minimize our loss function

Challenge: differentiating through fixed point $\frac{\partial z^*}{\partial(\cdot)}$

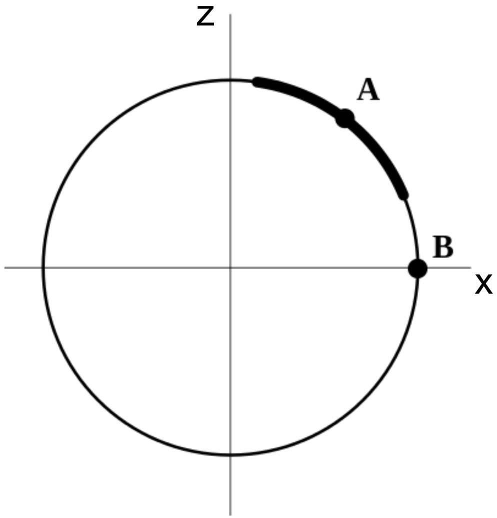
Naive Approach: Built-in Autodiff, through solver computation graph

- Memory Issues
- Floating Point Errors

Better Approach: Use Implicit Function Theorem!

Implicit Function Theorem (Informal)

$$f(x, z) = x^2 + z^2 - 1 = 0$$



Let f be a relation with inputs x_0, z_0

1. $f(x_0, z_0) = 0$
2. f is continuously differentiable with non-singular Jacobian

$$\partial_1 f(x_0, z_0) \in \mathbb{R}^{n \times n}$$

Then there exist neighborhoods (open sets) S_{x_0}, S_{z_0} around x_0 & z_0 and a function $z^* : S_{x_0} \to S_{z_0}$

1. $z_0 = z^*(x_0)$
2. $f(x, z^*(x)) = 0 \quad \forall x \in S_{x_0}$
3. z^* is differentiable on S_{x_0}

Image courtesy of Wikipedia

Implicit Function Theorem

High-level Idea:

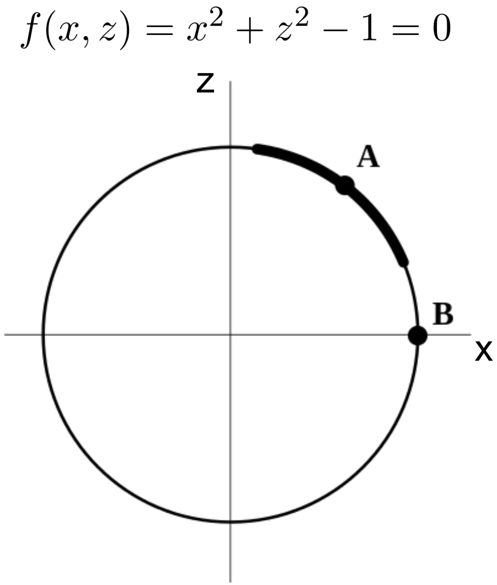
- Convert a relation to a function in a local region and find its derivative

$$f(x, z) = x^2 + z^2 - 1 = 0$$

- Explicit function at A:

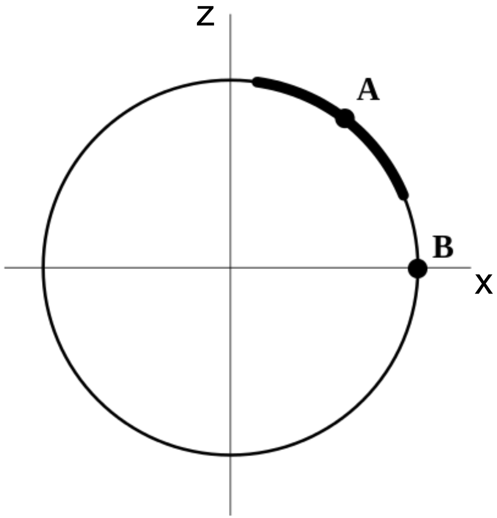
$$g_A(x) = \sqrt{1 - x^2}$$

- IFT allows us to find the derivative of *g*, *without* the explicit form



Implicit Function Theorem

$$f(x,z) = x^2 + z^2 - 1 = 0$$



Let $f : \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^n$ and $x_0 \in \mathbb{R}^p, z_0 \in \mathbb{R}^n$ be such that:

1. $f(x_0, z_0) = 0$
2. f is continuously differentiable with non-singular Jacobian $\partial_1 f(x_0, z_0) \in \mathbb{R}^{n \times n}$

Then there exist open sets $S_{x_0} \subset \mathbb{R}^p$ and $S_{z_0} \subset R^n$ Containing x_0 and z_0 respectively, and a unique continuous function $z^* : S_{x_0} \to S_{z_0}$ such that

1. $z_0 = z^*(x_0)$
2. $f(x, z^*(x)) = 0 \quad \forall x \in S_{x_0}$
3. z^* is differentiable on S_{x_0}

Backwards Pass

$$z_0 = f(x_0, z_0)$$

$$\Big\Downarrow \quad \text{By IFT}$$

$$z^*(x) = f(x, z^*(x)) \quad \forall x \in S_{x_0}$$

$$\boxed{\partial z^*(x_0)} = \partial_0 f(x_0, z_0) + \partial_1 f(x_0, z_0) \boxed{\partial z^*(x_0)}$$

$$\boxed{\partial z^*(x_0)} = [I - \partial_1 f(x_0, z_0)]^{-1} \partial_0 f(x_0, z_0).$$

Backwards Pass - DEQ

- **Backward Pass:**
  - Solve using root finding (e.g. Newtons)

$$\frac{\partial \ell}{\partial (\cdot)} = -\frac{\partial \ell}{\partial z^*}(J_{g_\theta}^{-1}|_{z^*})\frac{\partial f_\theta(z^*;x)}{\partial (\cdot)}$$

$$(J_{g_\theta}^\top|_{z^*})x^\top + (\frac{\partial \ell}{\partial z^*})^\top = 0$$

VJP easily obtained from Pytorch/Jax/etc.

$$g_\theta(z^*;x) = f_\theta(z^*;x) - z^*$$

Approximate Inverse Jacobian - Broyden's Method

- Expensive to calculate the inverse Jacobian during root finding for both forwards and backwards!

- **Broyden's Method (quasi-newton solver):**
    - During root finding, approximates the inverse Jacobian using

$$J_{g_\theta}^{-1}|_{z^{[i+1]}} \approx B_{g_\theta}^{[i+1]} = B_{g_\theta}^{[i]} + \frac{\Delta z^{[i+1]} - B_{g_\theta}^{[i]} \Delta g_\theta^{[i+1]}}{\Delta z^{[i+1]\top} B_{g_\theta}^{[i]} \Delta g_\theta^{[i+1]}} \Delta z^{[i+1]\top} B_{g_\theta}^{[i]}$$

Initial Guess: $B_{g_\theta}^{[0]} = -I$

$$g_\theta(z^*; x) = f_\theta(z^*; x) - z^*$$

DEQ Memory

- **Very memory efficient** because forward and backward passes just use root-finding algorithms.
- Avoids over all the overhead from uncurling backpropagating steps.
- Storage:
  - Equilibrium Point z^*
  - Network Input x
  - Model f_θ
  - VJP (*no Jacobian construction needed!)

Expressivity of DEQs

Intuition:

Consider a simple function composition $$y = g_2(g_1(x))$$
Transforming this into a DEQ:

$$f(z, x) = f\left(\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}, x\right) = \begin{bmatrix} g_1(x) \\ g_2(z_1) \end{bmatrix}$$

Thus, the equilibrium point is:

$$z^\star = f(z^\star, x) \iff z_1^\star = g_1(x),\ z_2^\star = g_2(z_1^\star) = g_2(g_1(x))$$

The output equilibrium is the output of the function!
(can be extended to arbitrary computation graph)

DEQ Stacking?

- Stacking DEQs don't really work, as a single DEQ layer can model any amount of stacked DEQ layers.

Intuition:

Consider a stack of 2 DEQ layers:

$$z_1^\star = f_1(z_1^\star, x) \longrightarrow z_2^\star = f(z_2^\star, z_1^\star)$$

This is equivalent to the following single equilibrium problem:

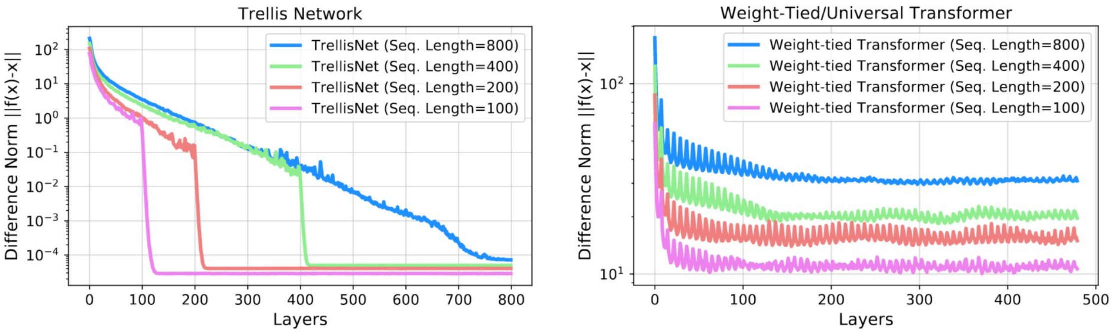
$$z^\star = \begin{bmatrix} z_1^\star \\ z_2^\star \end{bmatrix} = \begin{bmatrix} f_1(z^\star, x) \\ f_2(z_2^\star, z_1^\star) \end{bmatrix} = f(z^\star, x)$$

Experiments

- Apply Deep Equilibrium Networks to sequence modelling tasks
    - Sequence empirically converge
    - Already use of weight tying over the temporal sequence (Trellis Nets & Transformer models)
    - Long-range copy-memory, Penn Treebank Language Modelling, WikiText-103
- Demonstrate the memory efficiency and expressivity of DEQ models given similar parameter counts as well as the speed of computation

Convergence Caveat

- One might expect the network to diverge in the infinite limit
- In practice, many networks do not, which is explored more formally in a later work [Kolter et. al 2020]
- In this work, the authors empirically show that the contributions of subsequent layers diminish at very large depths



Experiments

Word-level Language Modeling w/ WikiText-103 (WT103)				
Model	# Params	Non-Embedding Model Size	Test perplexity	Memory†
Generic TCN [7]	150M	34M	45.2	-
Gated Linear ConvNet [17]	230M	-	37.2	-
AWD-QRNN [33]	159M	51M	33.0	7.1GB
Relational Memory Core [40]	195M	60M	31.6	-
Transformer-XL (X-large, adaptive embed., on TPU) [16]	257M	224M	**18.7**	12.0GB
70-layer TrellisNet (+ auxiliary loss, etc.) [8]	180M	45M	29.2	24.7GB
70-layer TrellisNet with *gradient checkpointing*	180M	45M	29.2	5.2GB
DEQ-TrellisNet (ours)	180M	45M	**29.0**	**3.3GB**
Transformer-XL (medium, 16 layers)	165M	44M	24.3	8.5GB
DEQ-Transformer (medium, ours).	172M	43M	24.2	**2.7GB**
Transformer-XL (medium, 18 layers, adaptive embed.)	110M	72M	23.6	9.0GB
DEQ-Transformer (medium, adaptive embed., ours)	110M	70M	**23.2**	3.7GB
Transformer-XL (small, 4 layers)	139M	4.9M	35.8	4.8GB
Transformer-XL (small, weight-tied 16 layers)	138M	4.5M	34.9	6.8GB
DEQ-Transformer (small, ours).	138M	4.5M	**32.4**	**1.1GB**

Not SOTA, but good at same param size

Efficient!

Experiments - Runtime

Table 4: Runtime ratios between DEQs and corresponding deep networks at training and inference ($> 1\times$ implies DEQ is slower). The ratios are benchmarked on WikiText-103.

DEQ / 18-layer Transformer		DEQ / 70-layer TrellisNet	
Training	Inference	Training	Inference
$2.82\times$	$1.76\times$	$2.40\times$	$1.64\times$

- Notice DEQs are slower!
- This is a consequence of solving an inner optimization inside the network

Conclusion

- Deep Equilibrium Models are a weight tied, approximately infinite depth neural network
  - Output is the fixed point of some neural network function
- Computes two root finding solutions for both the forward and backwards pass
  - Uses IFT to compute the gradient updates rather than backpropagation and autodiff through the iterative graph
- Performs comparatively to SOTA models of the same size but are considerably more memory efficient
  - Typically slower due to the inner loop optimization both forward and backwards

References

Deep Implicit Layers Tutorial - http://implicit-layers-tutorial.org/

Deep Equilibrium Models [Bai 2019] - https://arxiv.org/abs/1909.01377